GEO I: A COMPILER APPROACH TO MACHINE PROBLEM-SOLVING

Ronald Glenn Osborne

United States
Naval Postgraduate School

THESIS

GEO I: A COMPILER APPROACH
TO MACHINE PROBLEM-SOLVING

by

Ronald Glenn Osborne

Thesis Advisor: G. D. Gibbons

June 1971

Approved for public release; distribution unlimited.

GEO I: A Compiler Approach to Machine Problem-Solving

by

Ronald Glenn Osborne
Captain, United States Marine Corps
B.S., University of Mississippi, 1963

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1971

ABSTRACT

The work described herein should be viewed as experimental research in the area of machine problem-solving. The essence of such study is the utilization of a digital computer for the discovery of problem solution in regions which normally require the human faculty labelled intelligence. The domain of this effort was elementary Plane Geometry, including all of the assumptions, theorems and corollaries (and their associated exercises) normally considered in a first course. The ultimate goal was a machine which could attain a passing score on a final examination over the subject matter. The vehicle employed is a sizable computer program, designed and implemented under the facilities of a compiler generating system for execution on the IBM System 360.

TABLE OF CONTENTS

I. INTRODUCTION

The work described herein should be viewed as experimental research in the area of machine problem-solving. The essence of such study is the utilization of a digital computer for the discovery of problem solutions in regions which normally require the human faculty labelled intelligence. The domain of this effort was elementary Plane Geometry, including all of the assumptions, theorems and corollaries (and their associated exercises) normally considered in a first course. The ultimate goal was a machine which could attain a passing score on a final examination over the subject matter. The vehicle employed is a sizeable computer program, designed and implemented under the facilities of the XPL compiler generating system [1] for execution on the IBM System 360.

A. BACKGROUND AND MOTIVATION

Throughout the first decade of their existence, electronic computing machines were almost exclusively used as high-speed data processors. They performed the elementary operations of counting, adding and subtracting with amazing accuracy and did so at speeds which were considered phenomenal. Primarily for this speed and accuracy, in situations where human error was a constant factor, computers were popularly referred to as "electronic brains." Each individual within the computer profession, however, knew that there was no semblance of functioning in the machines comparable to that of the human brain. In fact, it

was difficult to program these devices for even the most fundamental data processing work.

Nonetheless, some individuals recognized the great potential to be explored and began searching for other means of using the capacities of this relatively new invention. Among the pioneers was A. M. Turing, an English mathematician and logician. In a paper published during 1950 [2], he considered the question "Can machines think?" The obvious answer was unmistakably negative if the definition of the word "think" was restricted to a process attributable only to human beings. Mainly to avoid semantic arguments and to provide a basis for further discussion, Turing proposed an alternate suggestion with the imitation game, best described by direct quotation from the paper:

> "The new form of the problem can be described in terms of a game which we call the 'imitation game.' It is played with three people, a man (A), a woman (B), and an interrogator (C) who may be of either sex. The interrogator stays in a room apart from the other two. The object of the game for the interrogator is to determine which of the other two is the man and which is the woman. He knows them by labels X and Y, and at the end of the game he says either 'X is A and Y is B' or 'X is B and Y is A.' The interrogator is allowed to put questions to A and B thus:
>
> C: Will X please tell me the length of his or her hair?
>
> Now suppose X is actually A, then A must answer. It is A's object in the game to try and cause C to make the wrong identification. His answer might therefore be:
>
> 'My hair is shingled, and the longest strands are about nine inches long.
>
> In order that tones of voice may not help the interrogator the answers should be written, or

better still, typewritten. The ideal arrangement is to have a teleprinter communicating between the two rooms. Alternatively the question and answers can be repeated by an intermediary. The object of the game for the third player (B) is to help the interrogator. The best strategy for her is probably to give truthful answers. She can add such things as 'I am the woman, don't listen to him!' to her answers, but it will avail nothing as the man can make similar remarks.

We now ask the question, 'What will happen when a machine takes the part of A in this game?' Will the interrogator decide wrongly as often when the game is played like this as he does when the game is played between a man and a woman? These questions replace our original, 'Can machines think?'"

By couching the problem to be discussed in the alternate form of the imitation game, Turing avoided prima facia objections to strike at the vital issues. If an electronic computer could satisfactorily participate in the game, then even the most adamant of its critics would concede that some simulation of human thought had been accomplished. If the computer were to participate and win a suitable number of times, those same critics could only admit that the machine had exhibited that intangible quality we call intelligence.

B. INTERVENING DEVELOPMENTS

During the 21 years which have passed since the first appearance of Turing's paper, a number of excellent works have been completed. Each is a study in the field of Artificial Intelligence, defined by Minsky [3] as

> "The science of making machines do things that would require intelligence if done by men."

Feigenbaum and Feldman [2] presented a collection of computer programs which included a chess player, a checker player, a

question-answering program, two problem-solving and two theorem proving machines. (One of the theorem proving machines, which lead directly to the present research, is briefly discussed later in this paper.) More recent works of note were those of Bobrow [4] and Evans [5]. Others are reported, in detail, by Minsky [3].

C. FORERUNNER OF THE CURRENT WORK

In a paper presented in 1959 [6], H. Gelernter described a project wherein the goal had been to design a machine whose behavior exhibited some characteristics of human intelligence. He concisely stated

> "The particular problem of theorem proving in plane geometry was selected as representative of a large class of difficult tasks which seemingly require ingenuity and intelligence for their successful completion."

Initial research revealed that Gelernter and his colleagues, J. R. Hansen and D. W. Loveland, had recounted their several years of labor in a number of additional reports [7, 8,9,10]. Careful study of these reports, in the chronological order of their appearance, provided a complete description of the project and simultaneously pointed to renewed research in the same area, based on the evolution of drastically improved equipment and techniques.

The original geometry machine, written in FORTRAN, comprised some 20,000 individual instructions [8]. Although ill-suited to any list-processing situation, FORTRAN was the language of the day. Further, its employment forced the development of a FORTRAN-compiled list processing language

[9], a non-trivial task in itself. Dynamic memory allocation concepts were totally lacking so the machine employed a scheme devised by Gelernter [10]. Finally, the equipment available to Gelernter and his associates included an IBM 704 computer with its several peripheral units. The System 360, with its voluminous memory capacity and high-speed direct access storage devices, was unknown.

D. OVERVIEW OF THE PRESENT SYSTEM

Contemplation of the various aspects of the original project from the vantage point of today's equipment and technology offered the opportunity for a fresh attack and the promise of major improvements. Several scattered ideas became outlines and an entire system began to take form in terms of a modern language. This new system retained the basic goal of developing a machine which exhibited some characteristics of human intelligence. The achievement of a passing mark on a final examination in Plane Geometry was established as the unquestionable criterion of success.

Further study produced a list of desirable features to be included as the work progressed toward its goal. Each one mentioned below is treated separately, with examples, in the discussion of methods (Section III).

1. Input

Through a specific grammar, tailored to Plane Geometry, a restricted but adequate dialect is accepted. Exercises to be completed by the machine are stated in a language similar to that used in modern textbooks [11,12].

2. Compilation

GEO I uniquely employs several concepts normally associated with compiler systems, in the construction of an internal problem representation. Incoming information is transformed into a data structure which is stored in a dynamically-allocated scratchpad memory section.

3. Analysis

Completion of the input and data-storing phases marks the beginning of the problem analysis. Procedures within the system examine the known information in terms of the desired goal (i.e. the stated requirement of the exercise or a related step).

4. Resolution

The performing unit, upon receiving guidance from the analysis section, attempts to satisfy the topmost goal of a "goal stack." Establishment of either success or failure returns control to the analysis unit for additional guidance or ultimate completion, as appropriate. Subgoals are generated and added to the pushdown store, if required.

5. Output

The design of GEO I provides an intermediate trace of its various attempts in case no solution is discovered. Satisfaction of the last goal (the primary objective) on the pushdown store results in the statement of the input problem followed by the individual steps of the proof in an argument-defense format.

II. CONSIDERATIONS OF INTELLIGENCE

Any simulation of human thought processes presupposes an understanding of those processes commensurate with the scope of the simulation. However, such an understanding is not readily acquired. Men have devoted their entire professional lives to that effort alone; yet, few individuals lay claim to any profound knowledge. Volumes have been written which provide varying degrees of insight; still, the functional peculiarities of the human brain lie largely unexplained. In reality then, preparation for a machine simulation of human thought processes is a difficult exercise in observation and speculation: observation of behavioral characteristics and speculation about their causes.

Information must be amassed to answer the many questions which arise in the later stages. The individuals desiring to perform the simulation inspect the behavior of other individuals (and their own behavior as well) in the environment under consideration. The literature may provide the results of additional efforts in the same or related areas. Regardless of the source, each result must be carefully considered for its validity and applicability before it becomes an element of the information store.

Men, after observing other men, can only state a probable cause for the behavior just witnessed. Individuals are rarely able to explain fully their own thoughts as they solve an

elementary puzzle. Some indeterminate amount of speculation is inherent in either case. This may be reduced by consulting more informed sources, but it cannot be eliminated. Absence of absolute facts regarding thought processes must be reckoned with throughout the machine simulation.

A. HUMANISTIC APPROACH

Pertinent considerations in the various regions of the human problem-solving environment are well-established, despite the fact that relatively little information is known about thought processes. The particular history of Plane Geometry originates with the Egyptians, prior to 1850 B.C. [12], although Euclid is accredited with the first formal treatment about 300 B.C. [13]. Since those early beginnings, men have questioned not merely if they were able to solve a problem, but how they solved it as well. Such curiosity lead to investigation and eventually, to documentation. Further investigation resulted in refinement of the written work and so on and on. Today, even the most elementary problem text suggests a list of steps to follow in the search for a solution.

Notable among contemporary works on problem-solving are the five volumes by the mathematician, G. Polya [14,15,16,17, 18]. His contributions reflect his many years of work in the field, both as a student and as a professor in a major university. (The five volumes were written over the twenty-year period: 1945-1965.) Although other works were examined in

the search for additional approaches, none were found to equal those named above.

Polya presents a basic method of attack [14] comprising four distinct phases. Each of fifteen minor steps offers questions or suggestions contrived to lead the student along a potential solution path.

1. Understand the Problem

Rarely is a first reading adequate for understanding a mathematical exercise. Occasional references to appendices or glossaries are needed to locate definitions of new or temporarily forgotten words. When the requirements are presented in word problem format, several readings may be necessary, focusing attention on a different small segment each time. Rephrasing the problem may remove semantic complexities which obscure the specific issues. Polya poses the following:

"a. What is the unknown? What are the data? What is the condition?
b. Is it possible to satisfy the condition? Is the condition sufficient to determine the unknown? Or is it insufficient? Or redundant? Or contradictory?
c. Draw a figure. Introduce suitable notation.
d. Separate the various parts of the condition. Can you write them down?" [14].

2. Devise a Plan

This single step is often the difference between successful completion and total failure of the overall effort. Without a sound plan of attack, the student will wander hopelessly about in a prodigous solution space. He must find some connection between the data and the unknown.

Close examination of the known information, complemented by previously learned concepts and related experiences, will frequently tend to provoke an appropriate idea. Some thought should be given to ancillary solutions which could enable the primary result. Useless information is seldom given. Every effort should be made to employ all that is presented.

"a. Have you seen it before? Or have you seen the same problem in a slightly different form?
b. Do you know a related problem? Do you know a theorem that could be useful?
c. Look at the unknown! And try to think of a familiar problem having the same or a similar unknown.
d. Here is a problem related to yours and solved before. Could you use it? Could you use its result? Could you use its method? Should you introduce some auxiliary element in order to make its use possible?
e. Could you restate the problem? Could you restate it still differently? Go back to definitions.
f. If you cannot solve the proposed problem try to solve first some related problem. Could you imagine a more accessible related problem? A more general problem? A more special problem? An analogous problem? Could you solve a part of the problem? Keep only a part of the condition, drop the other part; how far is the unknown then determined, how can it vary? Could you derive something useful from the data? Could you think of other data appropriate to determine the unknown? Could you change the unknown or the data, or both if necessary, so that the new unknown and the new data are nearer to each other?
g. Did you use all the data? Did you use the whole condition? Have you taken into account all essential notions involved in the problems?" [14].

3. Execute the Plan

Heavy emphasis should be placed on a stepwise progression toward the solution, using the devised plan of attack as a guide. Each statement, each small result should

be directly verifiable from previously derived results or given facts. Previously derived results include past theorems, corollaries and definitions as well as those statements formulated during the present attempt.

4. Examine the Solution

Apparent successful completion should be followed by two final steps:

a. Evaluate the Accuracy

Arriving at a solution via the recommended method offers indisputable accuracy, if each individual step has been correctly derived. However, if the result can be verified by another means, that means should be applied.

b. Evaluate Potential Application

Association is a valuable asset in the attack-planning process. Similarities among problems often suggest the required plan. Thus, each salient characteristic of the exercise just completed should be examined for future application.

Routinely proceeding through the above outline does not guarantee the appearance of a solution as an end result. It must be accompanied by some previous knowledge of the subject matter. (Additionally, the student must have a desire to solve the problem, an ability to read and some means of communication. These are presumed.)

One final point should be included. The consideration of ancillary solutions, mentioned during the discussion of phase two, deserved additional amplification. Once the ancillary problem is formulated, it should be subjected to

the entire four-phase procedure, just as though it were the only exercise to be performed. One can readily see a cycling process necessary to derive more complicated results. Once the first step is complete (i.e. understand the problem), the main effort may be subdivided into number of smaller segments, each demanding attention. The primary effort may be completed only after the separate pieces have been properly assembled.

B. CONTRAST OF METHODS

Each subject which lends itself to the application of particular problem-solving procedures will also include the possible question of a large number of sequences which do not represent the solution. For example, consider the class of all possible exercises in Plane Geometry. For each problem within this class, one may properly derive many true statements which do not approach the final solution. Further, if all of the necessary steps are included in the list, they may not be ordered in a manner which constitutes an appropriate proof. Obviously, some means must be devised which eliminates such futile meandering.

1. Algorithms

One possible method is an algorithmic procedure, which guarantees a correct solution, if a solution is discovered. Although certain specific qualifications determine whether or not a procedure is an algorithm, they are not essential to the desired understanding here. Simplistically, an algorithm may be viewed as a sequence of steps which solve a particular type of problem. The given information is applied

according to regid instructions and the required operations are performed (again, according to rigid instructions). If this process yields any solution at all (some algorithms do not) then its correctness is guaranteed.

It should be obvious, however, that such processes are severely limited, both in the classes of problems for which they exist and in the amount of intelligence which they require for successful operation. There are many more interesting problems than sufficient algorithms.

2. Heuristics

Of much more interest are the heuristic methods, which consider only a portion of the possibilities. Although a number of semantically different definitions exist, the one presented by Gelernter and Rochester is the one adopted for this work:

> "...a heuristic method is a procedure that may lead us by a short cut to the goal we seek or it may lead us down a blind alley." [7]

This method is not guaranteed to find a solution. Heuristic methods may, in fact, overlook an existing solution, by definition. A bad heuristic will yield many such cases while an excellent method will overlook relatively few.

3. Contrasts

Both methods have their advantages and their place. The algorithmic method is superior for its guarantee of producing a solution. Heuristic methods may be applied to any class of problems and have the potential of a short cut. The value of the short cut is commonly highlighted with the example of a chess game. Theoretically, chess could be played by

an algorithmic procedure which evaluates all possible moves for each piece at each play. However, Shannon [2] estimates that a single computer would be forced to evaluate 10^{120} paths for a single game, thus requiring an impossibly long time. A heuristic system must be employed which eliminates many of the choices from consideration in favor of the more potent ones.

Heuristic methods for problem-solving situations have been refined over hundreds of years. Many procedures work so well, particularly when coupled with human intelligence and ingenuity, that they nearly qualify as algorithms in the sense of guaranteeing a solution. Yet, there exists that special case or a particular trick which will not permit such a classification.

C. MACHINE CONSIDERATIONS

Close inspection of human behavior allows analogies to be drawn in the computer simulation of this behavior. The desire is to provide the machine with the same information and procedures available to the human and to receive the same result from each. If the tasks presented to each are sufficient in number and difficulty and if the results achieved are equivalent, then the machine has, in some manner, exhibited intelligence.

The chess example above stands as sufficient testimony to the requirement for heuristic methods, in the absence of better algorithms. The inherent speed of the electronic machine will not overcome the obstacles presented in difficult problem

classes. One immediately turns, then, to a heuristic method and attempts to forge its likeness into a computer program.

Such was the case with the development of GEO I. Plane Geometry problems do not readily lend themselves to an algorithmic approach. Consequently, the four phase method of Polya [14] was adopted and modified. This modified version, containing essentially the same steps, was then implemented, (Discussion of the actual implementation is provided in Section III. The remaining paragraphs here are presented only for analogy purposes.)

1. Understand the Problem

General programming languages provide a medium for man-machine communications. A person, desiring to solve a problem by machine, provides a list of instructions written in some language which the machine understands. Such understanding is a complex mass of electronic switching logic functionally controlled by the digit sequences of the instructions. (See Section I.) When the programming languages for a given machine are inadequate, the individual may write one of his own, add a translatory device and communicate via the new language. In any case, the capacity to conduct two-way communications must be present. This capacity is assumed in the humanistic areas of problem-solving but adequate internal representation presents a considerable obstacle in the simulation process.

2. Analyze the Known Information

At this point, an additional step was added during the modifications mentioned above. Polya [14], assuming the

student's ability to read and write, proceeds directly with analysis of the various conditions and information. Such analysis within GEO I, as with most problem-solving programs, is performed by a separate procedure.

Particular note was made of his suggestion for a figure. Diagrams provide immeasurable assistance in solving Plane Geometry problems. (Perhaps even more assistance than in any other field of mathematics.) The data structure of GEO I (See Section III-D) serves the purposes intended by Polya.

3. Devise a Plan

This particular step possesses the same gravity in the computer application as it does with the human process. The plan must be devised carefully, avoiding all pitfalls. The speed of the machine will overcome an amount of inefficiency but it must not be relied upon. The human mind appears to readily skip from one idea to another if the initial effort is not productive. The machine, on the other hand, must have guidance to preclude its pursuing an exhaustive, fruitless search for conditions.

A number of guidance methods are employed. One of the more common uses flags (specific variables) which are set at various stages of operation. Another method is the iteration counter which simply increments a particular variable each time an operation is performed. A third technique, less frequently used, is the timer. The common denominator utilized is an upper limit, which, when detected, causes a return of control to the main effort.

At this point, Polya [14] suggests consideration of an auxiliary problem. Machines can be programmed to do the same. GEO I employs a pushdown store which is nothing more than a list of tasks to be accomplished. An observed need for a specific piece of information creates a new task which, by design, is immediately processed.

No effort is made to employ previous problems to current exercises. The machine does not learn by experience.

4. Carry Out the Plan

If the human derives each step correctly in proceeding toward the solution, the accuracy is assured. If the programmer has performed his job correctly, in every detail, then the accuracy of the machine solution is also guaranteed. However, any desired checking procedures must be built into the program. The machine must be instructed on checking, just as it was on finding the initial solution.

III. IMPLEMENTATION

The fundamental capacities of GEO I were dictated by the Minsky definition of Artificial Intelligence [3]. To satisfy this definition, the machine would be required to accept an English problem statement, solve the problem and communicate its results. Necessary abilities within the problem-solving process were determined by analyzing this process in some detail (as just presented in the previous section). In order to solve the exercise, it would be necessary to convert the incoming English statements into a flexible, internal representation of the problem. Performance of some elementary analysis of the data collection could immediately begin. Some general procedure was obviously required to derive the solution. Finally, the nature of Plane Geometry problems prescribed a form of heuristic control.

The general necessities were next examined in the light of machine and language requirements. Total storage and run time requirements were approximated for several different languages. The instruction sets of languages were examined for particular operational capabilities. For example, FORTRAN was immediately eliminated for its inadequate string-processing capabilities. PL/I programs typically require more time in the compilation step than programs, written in XPL, which perform the same function. LISP and SNOBOL suffered the same time disadvantage. Further, only XPL offered the potential compiler system application discussed below [1].

A. XPL SYSTEM

XPL is a programming dialect of PL/I, devised by McKeeman, Horning and Wortman. The XPL system is described by the authors as a translator writing system for the IBM System 360, expressly designed to permit a translator first to be written and then turned into code. (For a general discussion of compiler-translators, see Section III-C.) Simplistically viewed, the XPL system is a compiler-generator which permits the user to construct his own programming language and write the necessary compiler. Although the system contains several major components, the primary ones are ANALYZER, SKELETON and XCOM.

The ANALYZER accepts the newly-designed language, written in Backus-Naur Form [19], and translates it into syntax tables. SKELETON is a proto-compiler containing several utility routines and the mandatory parsing procedures. Equipped with the syntax tables produced by the ANALYZER, and appropriate instructions accompanying each production, SKELETON becomes the compiler of the new language. It may be subsequently compiled by XCOM, the master compiler for the language. Programs written in the new language are, in turn, compiled by the newly developed compiler, utilizing the coding instructions there.

1. Salient Features

The complete XPL system offered a number of potential advantages to the particular case of GEO I. Perhaps the greatest of all was an opportunity for the novel employment of a

compiler in a problem-solving environment (see Section III-C). Another distinct asset was the freedom permitted in the development of the grammar for Plane Geometry (see Section III-B), essential to the rules of the imitation game. Further, the availability of the SKELETON compiler allowed attention to immediately focus on the particular instructions required by the productions in this application. Finally, the various instructions contributed much to the efficient processing demanded by GEO I.

a. DO CASE Construct

Essentially a dispatcher, the DO CASE construct yields an efficient multi-branch decision capability, contingent on the preset value of a variable. For example, DO CASE (I) is executed as an unconditional branch to the I-th set of subordinate instructions. Moreover, this I-th set may comprise a single command, a group of instructions or still another DO CASE construct. This feature was of value in the GEO I machine, which frequently required the selection of a single branch from many alternatives.

b. Word Manipulation

Efficient use of internal storage is provided by the several instructions which rapidly access and modify individual System 360 words. (A word is made up of four bytes, each containing eight binary digits called bits.) Information may be packed and unpacked with shift and Boolean logic instructions.

c. String Manipulation

Character strings may be constructed, dissected, reassembled and transformed with a small group of XPL instructions. Substrings and string length are derived with single commands. String comparisons can be performed directly. The assembly process is performed by the catenate operator "||".

d. Storage Methods

Rigid data structures are not necessitated by the usual applications of translatory systems; hence, none are provided within the XPL complex. This serves as an advantage to the user, in that he may introduce a structure tailored to his implementation. It does, however, have the marked disadvantage of demanding time and effort for its implementation.

2. Deficiencies

Despite the observed cautions of the planning stages, some limitations and inherent disadvantages were unforseen. Absence of the data structure was just one example. Others were encountered, addressed and eventually traversed.

a. String Limitations

Basic to the GEO I concept was the idea of string manipulation. The simulation presumed access to a complete textbook for Plane Geometry. XPL limits the entire set of string descriptors to 4096 bytes. Initially considered adequate, this limitation was actually severe, forcing a number of sweeping alterations to the general design of GEO I.

b. Procedural Restrictions

The one-pass concept of XCOM (see Section III-C) produces two additional requirements. Again, the study

conducted prior to the selection of a language had discovered them and again, they were deemed acceptable.

(1) Ordering. Each procedure in the program must be physically located in the program deck before calls to the procedure. This restriction entailed some manipulation where interrelated blocks were concerned. The convenience of being able to call any procedure from any point outside that procedure was sacrificed to the advantages of the one-pass compiler.

(2) Non-Recursive Procedures. XPL procedures may not be recursively called. This disadvantage was subjected to particular scrutiny for its impact on GEO I and initially discounted as inconsequential. Its significance was not realized until development of the problem-solving procedures was well under way.

B. THE BNF DESCRIPTION OF PLANE GEOMETRY

The first requirement in the development of GEO I was the construction of an unambigous phrase-structure grammar (see Appendix A). This grammar was written in BNF (BACKUS-NAUR FORM [19] to satisfy the demands of the XPL translator, with primary emphasis on the content of the individual productions. (Both McKeeman [1] and Lee [20] offer excellent treatments of BNF and its application to formal grammar description for those unfamiliar with its employment.) Careful regard for the anticipated use of the productions during the problem input and data storing phases greatly simplified the later tasks of forming the data structure and analyzing the given information.

Secondary attention was devoted to the matters of completeness and generality to insure the capability of formulating any Plane Geometry exercise, using a language not unlike that of many textbooks. Additional consideration was given to the inclusion of graphic and time-sharing modes of operation and some productions were included which specifically served those objectives.

The original plans for GEO I included the visual display of a figure associated with each problem, as well as a time-sharing mode of operation for the problem-solving phase. Inclusion of either feature would have significantly altered the development of the entire system by providing a number of advantages over the batch mode of operation. However, operational considerations forced the deletion of both features. Availability of the graphic display unit was so severely limited that it precluded such a sizeable undertaking. Implementation of the XPL system under time-sharing did not materialize as forecast.

1. Creation of the Grammar

Perusal of problem statements in a number of textbooks revealed that the productions could be separated into the three principal categories discussed below. Further subdivision within each main section provided additional stages or levels essential to the data collection procedures. Continued application of this segmentation eventually provided sufficient individuality within each production to fulfill both the requirements of the XPL ANALYZER and the forecast needs of GEO I.

a. Production Classification by Task

Careful examination established only two basic tasks which confront Plane Geometry students, differing simply by the number of steps required to complete them. The first, which elicits only a single action is the command (e.g. draw line AB; join point A and point B; define "quadrilateral"). The second (and more interesting) type is the exercise, which requires a geometrical proof of several steps. The general classification "exercise" was further subdivided into three types; distinguishable by the disposition of their completed results.

(1) Theorems, if satisfactorily completed, were stored as usable tools in succeeding exercises.

(2) Corollaries were associated with the appropriate theorem, then admitted as tools for subsequent proofs.

(3) Practice problems were simply completed and destroyed (i.e. no attempt was made to preserve the results of problems for future machine reference).

b. Production Classification by Figure

Two basic figure types were identified: namely entities and structures. The former class was designed to include points, lines (straight and curved) and angles while the latter comprised all closed figures including all polygons and the circle. Productions within each class were then expanded to incorporate the various connotations which could be associated with each main type. For example, the figure described by the words LINE, ALTITUDE and MEDIAN is a straight line. However, ALTITUDE and MEDIAN are specific line types,

quite different in their effect on the problem-solving environment. Their definitions alone offer considerably more information than does the definition of "line".

c. Production Classification by Relation

Relationships were obviously necessary in order to state the most basic exercise. Four essential subdivisions were created to include all of the fundamental areas of problem solving.

(1) Comparison Relations: compare two elements in a quantitative manner (e.g. length, unit measure, equality).

(2) Locate Relations: spatial relationships necessary for graphic portrayal of the selected exercise such as above, left and adjacent.

(3) Conditional Relations: evoke specific figures or fundamental qualitative problem areas. Examples are vertical, parallel, colinear and intersect.

(4) Boolean Relations: logical and, or and not which were included only for possible employment in extensions of GEO I.

2. Testing and Refining the Grammar

Completion of the initial design phase of the grammar was followed by extensive testing with a variety of input specimens to ascertain which areas were complete and which areas required improvement. The grammar proved to be largely complete during the early runs. However, several significant modifications were made to annex desirable features.

a. The classification of string input was added to the TASK category. Strings are character strings, enclosed by single quotation marks, with only the restriction that the enclosed string may not contain a single quotation mark. They provided a very general form of privileged instruction, useful in modifying or correcting the textbook section of GEO I. For example, an input of 'THEOREM 63. THE EARTH IS FLAT.' replaced the existing THEOREM 63 with the new input string. String inputs were also useful in feeding the machine other information not of a problematic nature.

b. The structure <TRIANGLE> was separated from the general classification of polygons (closed figures), primarily because of its high frequency of appearance in Plane Geometry exercises. The author felt that problem analysis would be less complex if triangles were passed separately within the compiler.

c. The modification of greatest importance was the addition of the following productions:

```
<input> :: =    <input> *  <exercise>
                <input> *  <command>
                <input> *  <string>
```

This alteration permitted GEO I to accept sequential listings of tasks, limited by the XPL system to seventy-five total input cards per run. The potential (and likely) need to present GEO I with a number of sequential tasks in a single production run had been unforeseen in the original grammar.

d. The decision was made to include the XPL form of comment for amplification and clarity are required. (<comments> ::= /*<characters>*/ where characters may be any

EBCDIC characters except the */ combination). They are ignored by the XPL COMPILER and have no effect on the problem-solving capacities of the machine.

Throughout the implementation of GEO I, a close vigilance was maintained over the grammar with a view toward necessity of further alterations. Only a few minor innovations have been indicated and these, when viewed for their overall effect on the machine, have not been considered worthwhile.

Examples presented below illustrate the application of the BNF grammar to specific Plane Geometry tasks. Typical textbook formulation of a task is stated in each case followed immediately by the GEO I statement of the same problem providing for ease of comparison and contrast. Further examples are given in Section III-C including complete parsing.

FIGURE III-B1, COMMAND INPUTS (CONCATENATED)

TEXTBOOK SAMPLE:

1. DEFINE QUADRILATERAL
2. STATE THE DEFINITION OF PARALLELOGRAM.
3. WRITE THEOREM NUMBER 16.
4. BISECT LINE SEGMENT AB.

GEO I ACCEPTABLE INPUT:

```
 DEFINE QUADRILATERAL.
*STATE DEFN OF 'PARALLELOGRAM'.
*WRITE THEM 16.
*BISECT LINE AB.
```

The following examples are presented in the singular format for clarity. Catenation of exercises requires only an insulating asterisk between the last statement of one exercise and the first statement of the next:

(i.e. PROVE TRIANGLE ABC CONGRUENT TRIANGLE DEF.
•PROBLEM: GIVEN: LINE AC EQU LINE BF.).

FIGURE III-B2. EXERCISE INPUT EXAMPLES (SINGULAR)

EXAMPLE 1:

TEXTBOOK SAMPLE [11] :

GENERAL STATEMENT:
THEOREM 8. IF ONE OF TWO PARALLEL LINES IS PERPENDICULAR TO A THIRD LINE, THE OTHER IS ALSO PERPENDICULAR TO IT.

SPECIFIC STATEMENT:
GIVEN: TWO || LINES, AB AND CD; ONE OF THEM, AB ⊥ A THIRD LINE AC.
PROVE: CD ALSO IS ⊥ AC.

GEO I ACCEPTABLE INPUT:

```
THEOREM: GIVEN: LINE AB PARALLEL LINE CD.
                LINE AB PERPENDICULAR LINE AC.
         PROVE  LINE CD PERPENDICULAR LINE AC.
```

EXAMPLE 2:

TEXTBOOK SAMPLE [12] :

SPECIFIC STATEMENT:
GIVEN: X IS THE MID-POINT OF AB AND OF CD.
TO PROVE: CB=AD.

GEO I ACCEPTABLE INPUT:

```
PROBLEM: GIVEN: LINE AB WITH MIDPOINT X.
                LINE CD WITH MIDPOINT X.
         PROVE  LINE CB EQU LINE AD.
```

C. THE COMPILER UNIT

Compilers are normally designed as a matter of convenience for the user community of a particular computer. Machine language, accepted directly as input, is a complex mass of digits unlike any natural language. It is both confusing and cumbersome to the average user. Consequently, numerous higher level languages have been developed which allow users to

communicate their particular tasks to a machine through statements which more closely resemble natural conversation or written communication among human beings.

Addition of this convenience necessitated the inclusion of an intermediate processing phase betwen the user and the machine: one of converting one language to another by means of a large program. Such programs, distinguished by the type and the amount of converting which they perform, are variously referred to as translators, assemblers, compilers and interpreters. Lee [20] devotes his first chapter to a discussion of their basic differences. However, despite their established distinctions, each of the intermediate programs mentioned above shares some of its characteristics with the others.

1. Contrasts

The principal point of the below subsections is to contrast those common traits of compilers (as a general class) against the specific attributes of the related unit in GEO I. Figure III-C1 is a flow diagram which presents the major sections of the usual scheme while Figure III-C2 depicts the corresponding unit of the present machine. Reference to these figures should provide considerable assistance in understanding the several distinctions.

a. Input

Higher-level languages are designed in many cases, to satisfy the needs of a general segment of computer users. FORTRAN (FORmula TRANslator) is directed toward the scientific community while COBOL is directed toward the business world.

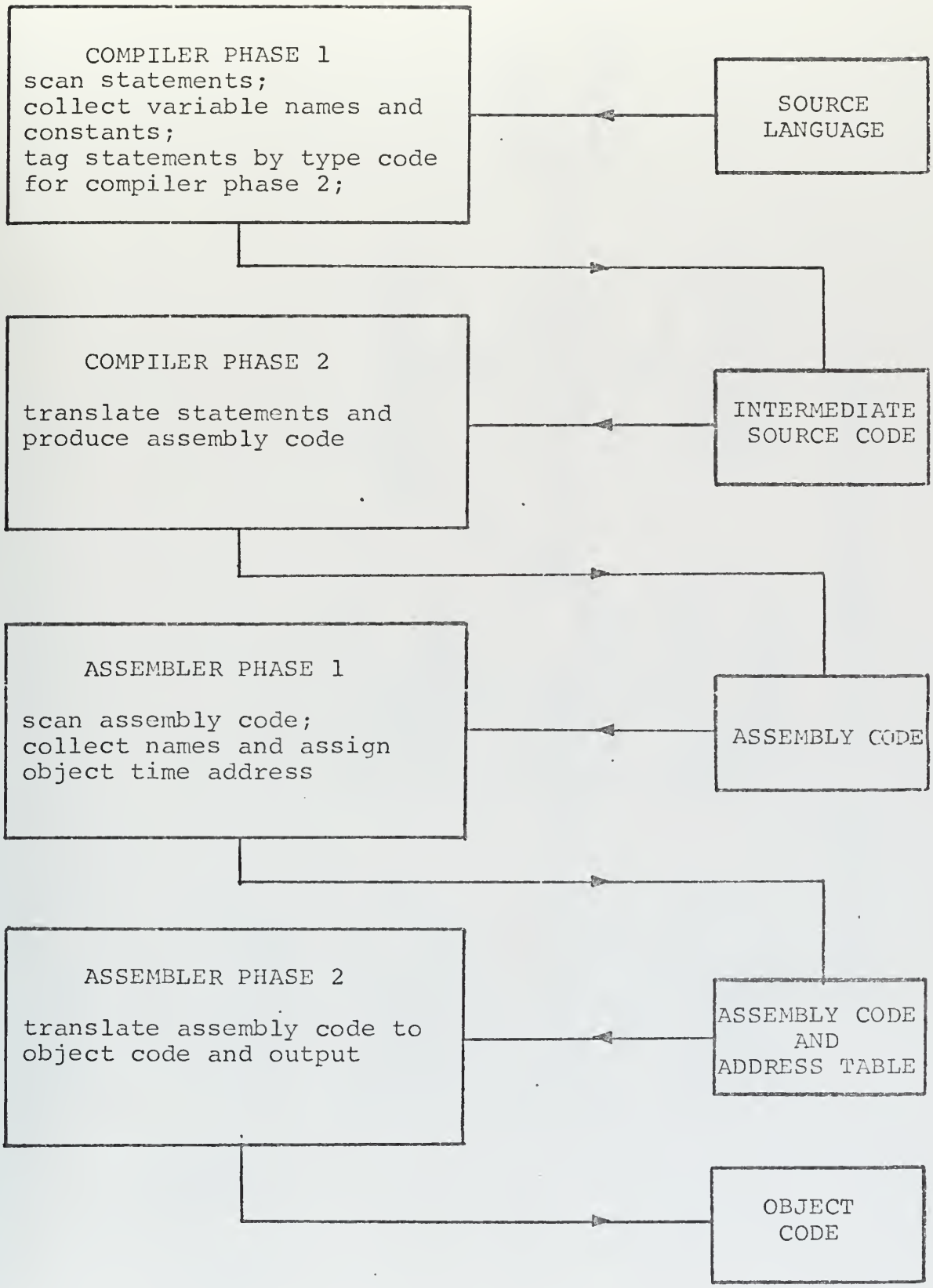


FIGURE III-C1. TYPICAL FOUR-PASS COMPILER [20]

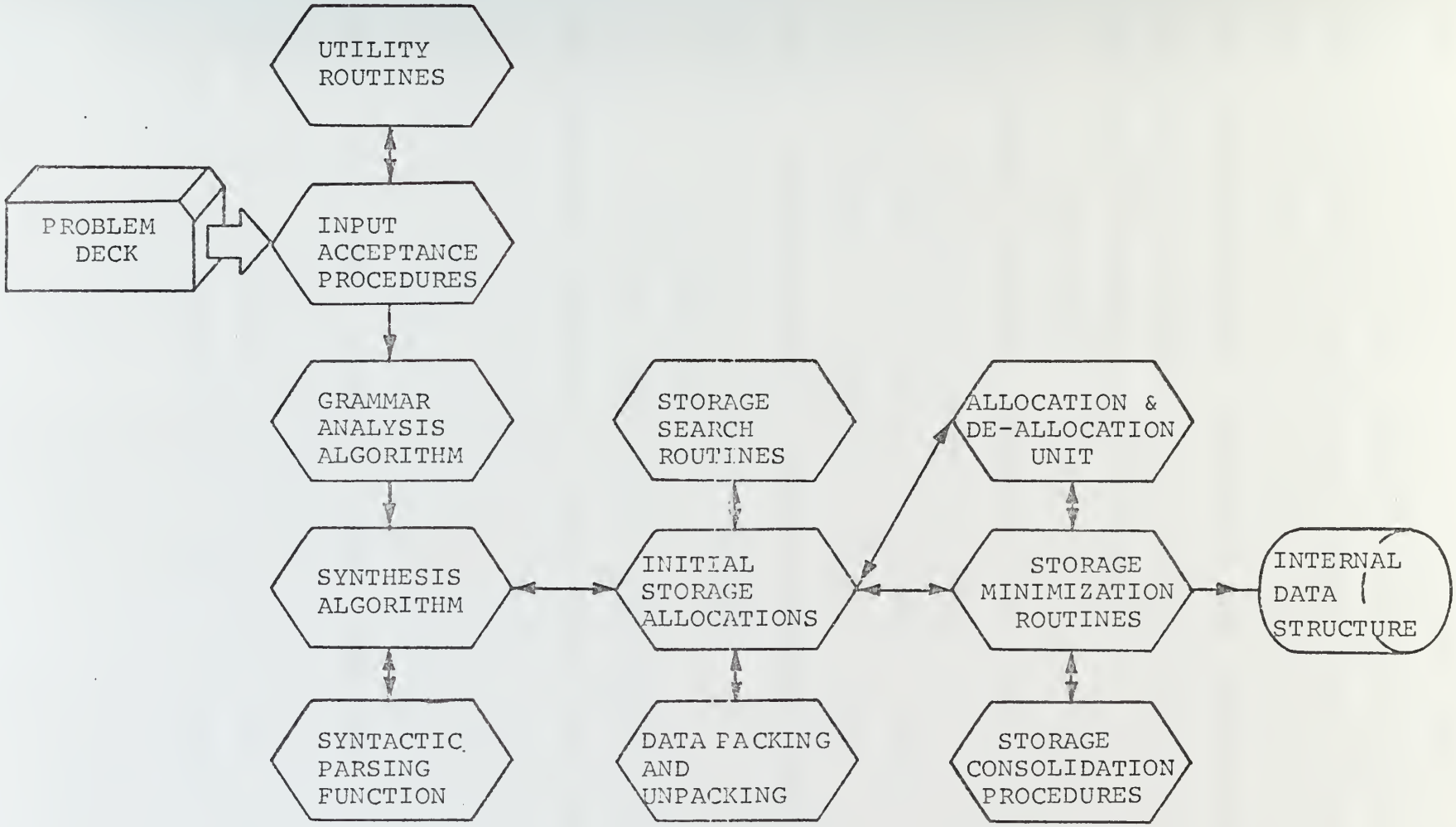


FIGURE III-C2. GEO I COMPILER FLOW DIAGRAM

SNOBOL is a string-processing language and GPSS is a simulation system. Yet their respective compilers exhibit a common characteristic in accepting the various input statements. Each individual job must comprise a complete set of instructions which meticulously prescribes the course of events. Every conditional branch, every arithmetic operation, every input-output statement must be specified.

An acceptable input to the GEO I unit, however, is the statement of either a command or an exercise (see Section III-B). The problem to be solved is presented to the machine in a format much like that of the geometry textbook. No specific guidance is offered on where to begin or how to proceed. Problem analysis must be performed internally before the solution sequence can be determined.

b. Transformation

In normal applications, the compiler transforms each individual statement into a sequence of instructions (i.e. strings of digits) which the computer may process during the execution phase. A particular statement type (e.g. DO I=1, 10) is subjected to the same analysis and is converted to the same sequence of digits each time it is encountered. Despite the complexity of the task created by nested loops, conditional branches and the like, nothing more is involved than employment of a very large program to convert one code to another.

By contrast, an input to GEO I is manipulated with regard for the context as well as the content. The various statements trigger and sustain the construction of a data

structure (see Section III-D) which is an internal reqre-sentation of the geometrical figures. Not one machine lang-uage instruction is created. Moreover, an attempt was made throughout the development of GEO I (in view of its ultimate goal: the imitation game) to create and maintain a represen-tation of the geometric meaning of each statement. Previously presented portions of the problem (within their respective data structures) are examined in the light of the new infor-mation of the current statement. These data structures vary as the information picture varies with the context of the incoming information.

c. Translation Stages

Many translatory systems in current applications complete their work only after iterative journeys (called passes) through the entire statement set. The majority of these employ three or four passes, converting a portion of the original code each time, although one popular compiler requires more than eighty passes. Such performance can be costly if total machine time is a major consideration.

The XPL system and its descendants make one pass through the input information, completing the necessary labors as they proceed. GEO I, in particular, does store the input cards internally but uses them only to print a copy of the entire problem on the same page as the proof. The problem-solving procedures rely totally on the internal data structure which has been established.

d. Passage of Control

Its conversion process at an end, the normal interpretive routine passes the resulting object program to other routines within the operating system. Actual execution of the instructions is ready to begin. In a like manner, control is passed by the compiler section of GEO I upon completion of the input process. But control remains within the GEO I system and execution is not immediately begun. Rather, the problem analyzer is called to perform its portion of the total effort.

2. Similarities

Despite its unique elements of departure from the customary employment of compilers, GEO I retained some general compiler concepts specifically to maximize their beneficial aspects. Chief among these concepts is canonical parsing: esentially a process of taking the input in small sections. Each section accepted is large enough to be distinct, yet small enough to permit frequent information-gathering cycles. Canonical parsing is generally employed in syntax-driven compilers. Due to its fundamental utility, the entire concept is explained here in some detail. Additional information is provided by McKeeman [1] in the early chapters of his book.

a. Canonical Parsing Algorithm

Succinctly stated, the task assigned to the parsing algorithm is the reciprocal of that performed by the programmer. In order to have the computer process a given job, one must determine what he wishes to say and formulate an

input which will be accepted by the compiler. Figure III-C3 shows a simple example.

FIGURE III-C3. CANONICAL PARSING EXAMPLE

ACCEPTABLE GRAMMAR:

```
<goal>        ::= <expression>
<expression>  ::= <term>
                | <expression> + <term>
<term>        ::= <primary>
                  <term> * <primary>
<primary>     ::=  X|Y|Z
```

DESIRED STATEMENT:

Form the sum of three variables named x, y and z.

PROPER PARSING:

```
                    <goal>
                      ↓
                <expression>
                      ↓
          <expression> + <term>
                      ↓
     <expression> + <term> + <primary>
                      ↓
          <term> + <primary> + Z
                      ↓
               <primary> + Y + Z
                      ↓
                  X + Y + Z
```

It should be noted that the particular grammar given expands only from right to left (i.e. Successive applications of productions three and five will expand a statement indefinitely to the left). The canonical parsing algorithm, receiving only the stream of input symbols, applies productions, working upward through a tree-like sturcture, to arrive at the goal symbol. This method is commonly referred to as "bottom-up" parsing. (Another general class exists but it is not applicable to either XPL or GEO I.)

(1) Stacking Decision Tables. One method of expanding the class of grammars acceptable to the parsing algorithm (employed in XPL) is the inclusion of a choice mechanism called a stacking decision table. The algorithm compares a number of symbols (two or less) on a stack with the next input symbol and selects from a three-valued function (stack, do not stack, conflict). Stack simply indicates that more information is required; do not stack yields an immediate canonical parse. Conflict means that a more involved comparison must be made in order to select a unique production.

As the BNF grammar is analyzed, the stacking decision tables may be constructed in the form of arrays, whose entries are the figures zero, one and two corresponding to the three functional values. The canonical parsing algorithm, provided with such tables, is then capable of accepting a much wider range of BNF grammars.

(2) Reduction Procedure. Employment of a scanning device permits the compiler to recognize the incoming symbols one at a time. References are made to both the stack and the decision table to determine the function value and the appropriate steps are performed. The value "do not stack" dictates a reduction. (Note that reduction, as used here, is a one-step move toward the goal symbol. It does not necessarily imply attaining a smaller parse.) The direction of this reduction process is solely dependent on the grammar and its particular productions. If the canonical form was expanded from right to left, the reduction proceeds from left to right and vice versa.

b. Canonical Parsing Algorithm for GEO I

Examination of the BNF grammar for Plane Geometry (see Appendix A) was performed with attention focused on the specific order of reductions to be performed. An understanding of this order provided numerous clues useful in establishing the data structure of GEO I and in determining the sequence of events throughout the problem-solving process. In particular, the grammar was observed to be left-recursive only in the production used for input catenation (<INPUT> * <EXERCISE>, <INPUT> * <COMMAND> and <INPUT> * <STRING>). All other rules which permit expansion are right-recursive.

Figure III-C4 presents the complete parsing of a single statement into its canonical sentential form to illustrate the value of understanding the algorithm. The order of parsing shows that the lines AB and CD are independently recognized, suggesting that a memory allocation (refer to the discussion of DYNAMIC MEMORY) take place at these points. Further, the completed statement parses into the canonical sentential form only after both lines and their connecting relationship (equality) have been identified. Observation of this sequence of events permitted the existing operation of the data structures to be established (i.e. storage of the structures and entities is completed before attempting to process any relationships).

Output from a number of computer runs assisted in further study of the parsing sequences which would be obtained in compiling exercises for GEO I. Only after this

INPUT STATEMENT: LINE AB EQU LINE CD.

PRODUCTION APPLIED:		RESULTING FORM:
<line type>	::= LINE	<line type> AB EQU LINE CD.
<strt seg>	::= <line type>	<strt seg> AB EQU LINE CD.
<line seg>	::= <strt seg>	<line seg> <identifier> EQU LINE CD.
<entity>	::= <line seg> <identifier>	<entity> EQU LINE CD.
<simple 1>	::= <entity>	<simple 1> EQU LINE CD.
<1 part>	::= <simple 1>	<1 part> EQU LINE CD.
<compare rel>	::= EQU	<1 part> <compare rel> LINE CD.
<relation>	::= <compare rel>	<1 part> <relation> LINE CD.
<line type>	::= LINE	<1 part> <relation> <line type> CD.
<strt seg>	::= <line type>	<1 part> <relation> <strt seg> CD.
<line seg>	::= <strt seg>	<1 part> <relation> <line seg> CD.
<entity>	::= <line seg> <identifier>	<1 part> <relation> <entity> .
<simple 1>	::= <entity>	<1 part> <relation> <simple 1> .
<1 part>	::= <simple 1>	<1 part> <relation> <1 part>.
<fact>	::= <1 part> <relation> <1 part>	<fact> .

FIGURE III-C4. GEO I STATEMENT PARSING

examination was essentially completed was it prudent to begin the work of writing the individual code sequences for the productions and establishing the data structure.

3. Code Development

Generation of appropriate processing sequences for acceptable inputs is a vital stage in the evolution of any compiler. Properly designed, these segments can insure efficient and accurate execution of each statement in the language. Improperly written, these same segments can yield gro-s inefficiency (hence, slower run times) and erroneous performance, if the system performs at all.

The final output of ANALYZER is a punched card deck consisting of various declarations and stacking decision tables required by the skeleton compiler. As an added convenience to the user, this deck also includes the productions of the grammar punched as individual XPL comments. The declarations and stacking tables are inserted directly into the forward portion of SKELETON. The individual production comments outline the synthesis procedure which will eventually become a mammoth DO CASE statement, with one case for each production in the grammar. (The grammar of GEO I contains 185 productions.) Each time a reduction is completed, the applicable production number is set and a call is emitted to the procedure SYNTHESIZE. Dispatching attention to the appropriate code steps is accomplished by the simple statement: DO CASE(PRODUCTION-NUMBER). The designer then begins the task of formulating the appropriate action to be taken for each case encountered. The proposed interaction between the

compiler section and the dynamic memory segments (see Section III-D) forced simultaneous development of both principle areas. The data structure was to be formed as the parsing progressed and required constant consideration.

D. DYNAMIC MEMORY

During the analysis of the human problem solving process, it became apparent that GEO I required a memory section analogous to the notepad or chalkboard, with provisions for all of the information associated with one particular exercise. The fact that processing of each exercise was to be terminated before attempting the next suggested the use of a dynamically-allocated work area. (Employment of such a scheme is common practice in large programs where total internal storage requirements are a consideration; reader familiarity with the general concept is assumed.) Five separate procedures comprising a complete system were adopted from a previous course in compiler writing [21].

Further contemplation of the storage requirements gave rise to the implementation of main and auxiliary (as required areas accessible to the individual problem. During the synthesis of a problem statement, structures and entities are examined in terms of their most elementary components (e.g. an angle is composed of three points, two lines and some quantitative measure, whether specified or implied, of the angle itself). Space is provided by the allocation system only if the incoming structure cannot be included as a substructure of some larger figure. Angle BAC will not be

provided with independent memory space if Triangle ABC has previously been stored. In the event that two or more distinct components of a more complex structure are stored prior to the recognition of the larger figure, the scheme combines the larger structure and the first of its previously stored components, returning the later space to the allocator.

Completion of the input for a particular exercise signals a procedure which explores the occupied scratch memory to minimize the overall requirements. A set of storage manipulation procedures, functioning as a unit, force the merger of blocks whenever feasible and release the extraneous memory. FIGURE III-D1 presents a typical problem input and offers a brief explanation of each step.

FIGURE III-D1

EXERCISE INPUT:

```
PROBLEM:  GIVEN:  ANGLE ABC EQU ANGLE DEF.
                  ANGLE BAC EQU ANGLE DFE.
                  LINE AB EQU LINE EF.
          PROVE   TRIANGLE BCA CONGRUENT TRIANGLE EFD.
```

STORAGE ALLOCATION:

Begin parsing of statement 1:

1. ANGLE ABC is assigned the prescribed amount of main storage.
2. ANGLE DEF is assigned the prescribed amount of main storage.
3. Indicators of the EQU relation are appropriately stored.

Begin parsing of statement 2:

1. ANGLE BAC is assigned the prescribed amount of main storage.
2. ANGLE DFE is assigned the prescribed amount of main storage.
3. Indicators of the EQU relation are appropriately stored.

Begin parsing of statement 3:

1. LINE AB is recognized as a component of ANGLE ABC. No storage allocation is required.
2. LINE DE is recognized as a component of ANGLE DEF. No storage allocation is required.

Begin parsing of statement 4:

1. ANGLE ABC is recognized as a component of TRIANGLE BCA.
2. The prescribed amount of main storage is allocated for TRIANGLE BCA.
3. Information stored under ANGLE ABC is copied into appropriate locations under TRIANGLE BCA.
4. Storage assigned to ANGLE ABC is returned to the allocator.
5. ANGLE DEF is recognized as a component of TRIANGLE EFD.
6. The prescribed amount of main storage is allocated for TRIANGLE EFD.
7. Information stored under ANGLE DEF is copies into appropriate locations under TRIANGLE EFD.
8. Storage assigned to ANGLE DEF is returned to the allocator.

Initial input is complete: the consolidation procedure is begun.

1. ANGLE BAC is recognized as a component of TRIANGLE BCA.
2. Information stored under ANGLE BAC is copied into appropriate locations under TRIANGLE BCA.
3. Storage assigned to ANGLE BAC is returned to the allocator.
4. ANGLE DFE is recognized as a component of TRIANGLE EFD.
5. Information stored under ANGLE DFE is copied into appropriate locations under TRIANGLE EFD.
6. Storage assigned to ANGLE DFE is returned to the allocator.

The problem analysis phase now begins. All information is stored under the triangles BCA and EFD; no additional storage is required.

FIGURE III-D1 (cont.)

1. Data Structure

Development of the specific data structure utilized by GEO I was tailored to the anticipated input with a guard to flexibility, workability and storage requirements. Memory was apportioned to the various geometrical figures on the basis of their complexity. A simple triangle, comprised of three angles, three lines and three points, requires substantially more memory than does a single angle. It should be noted, however, that the increase in assigned storage was not directly proportional to the increase in the number of sides or angles involved. (See Figure III-D2) Angles required forty-four words, basic triangles were alloted one hundred words and simple four-sided figures were provided with one-hundred twenty-eight memory words.

Encounters with more complex structures and special relationships dictated the development of auxiliary storage, not normally assigned but available if required. A quadrilateral in which both diagonals have been drawn contains no less than thirty-seven distinct problem components (e.g. six triangles, sixteen angles, ten line segments and five points). Special relationships or conditions include vertical angles, alternate interior angles and perpendicular lines. Incorporation of the auxiliary memory was essential to adequately provide for such complexities.

2. Data Storage

Ample space is provided for data storage within the framework allotted to each entity. During the input phase of

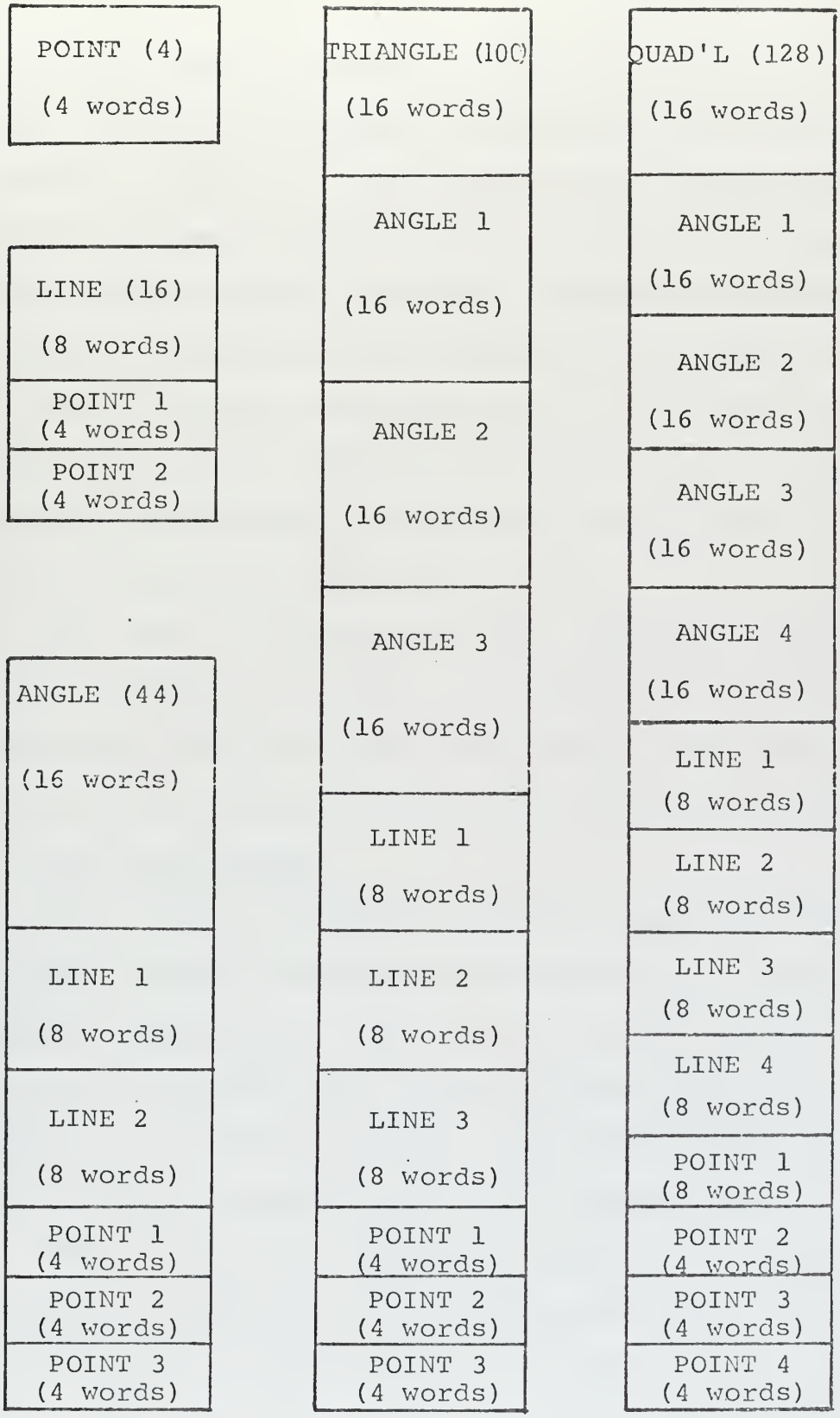


FIGURE III-D2. DYNAMIC MEMORY ALLOCATION

an exercise, the sundry elements are recognized as they are encountered although processing of relational data is begun only when one entire statement is complete (i.e. after parsing has reached a "<FACT>."). The appropriate location is then reached via the manipulation procedures and the information is packed into the space provided. A detailed discussion of the packed word formats is not material to the central theme of the problem-solving environment and is, therefore, not included. The point of interest is the capacity of rapid manipulation during both the input and problem-solving phases. Suffice it to say that each time an entire statement has been parsed, the given relationship and the appropriate identifier are cross-referenced in memory. For example, the statement "LINE AB EQUALS LINE CD" causes the relation "EQU" and the identifier CD to be stored under LINE AB and vice versa.

3. Data Manipulation

The policy of combining elemental substructures into larger ones created the fundamental necessity of efficient manipulation procedures. Close observation of the BNF grammar for the input revealed that a given problem might well result in numerous re-locations of data. The entire concept was carefully weighed, considering compact storage against access times, flexibility of the structure opposed to its rigidity and available machine time (i.e. turnaround time) for different core requirements. Other methods of storage and search were examined. The final decision was to continue with the original concept.

Dissection and tedious analysis of the manipulation task resulted in the creation of four distinct yet interwoven blocks of procedures which functioned as a single unit. (Simultaneous reference to Figure III-D3 will improve comprehension of the manipulation system.)

a. The first block was designed to provide checking and initial storage allocating facilities. During the problem synthesis, each entity is examined as it is recognized by the compiler section. As mentioned earlier, new section of memory are provided to only those structures which cannot be combined with existing segments.

b. Storage search procedures were required to locate specific substructures within the memory. They are utilized during the input phase and later in the problem analysis phase.

c. Movement of data from one storage location to another demanded the block referred to as procedures. As the complexities of the figures in an exercise becomes evident, information may be relocated to minimize the number of memory words occupied.

d. Data packing and unpacking are necessary at various points throughout the problem solving process. Specific word formats were designed to provide adequate information, minimal retrieval effort and minimal storage requirements.

E. PROBLEM ANALYSIS AND SOLUTION

The heuristics of GEO I are controlled by a network of procedures, functioning as a unit. Herein lie the machine steps for analyzing the information presented, in terms of

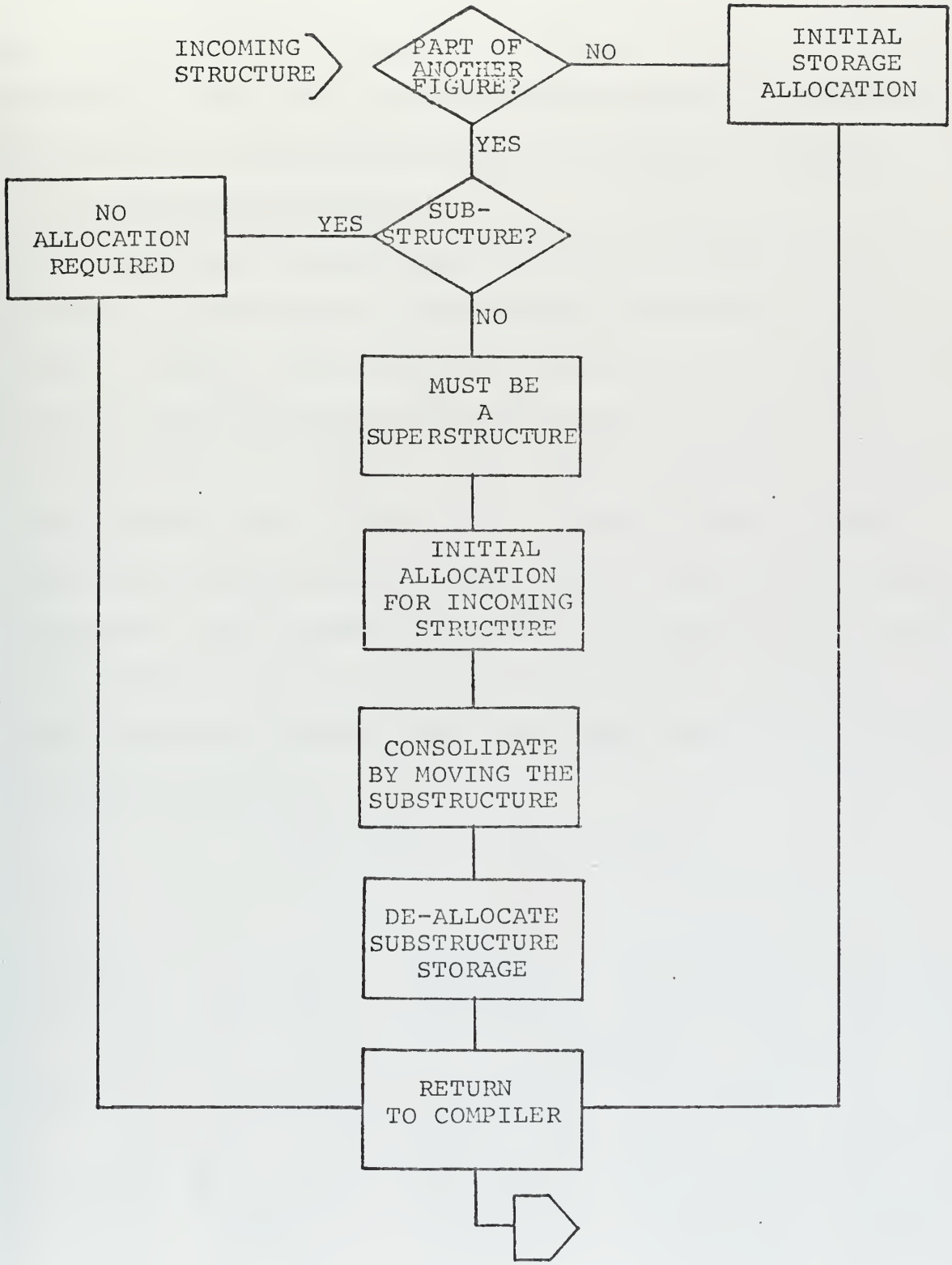


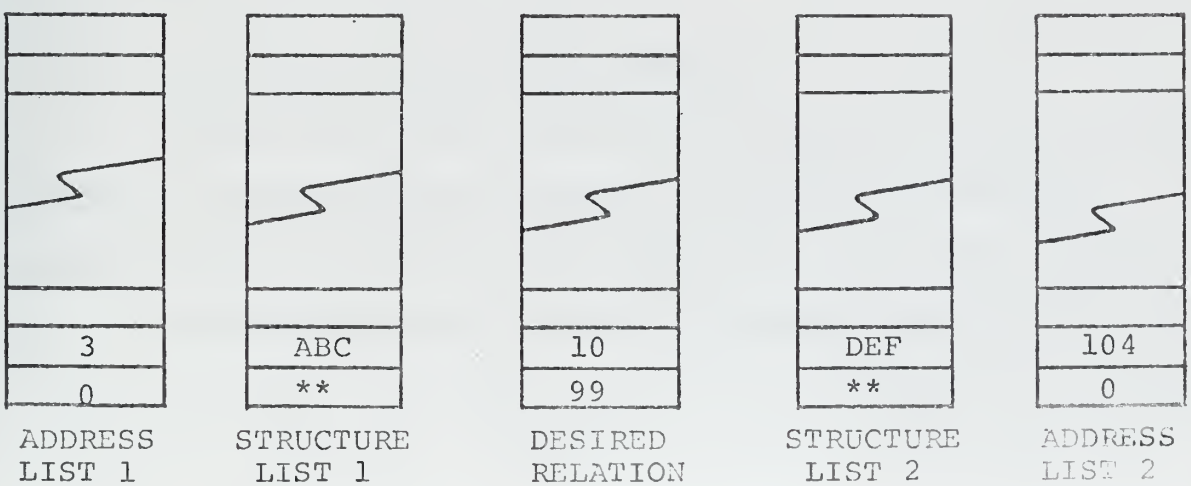
FIGURE III-D3. STORAGE MANIPULATION

the desired result, and for planning the attack. A single procedure, PREDICTOR, is the master planning device delegated the responsibility of overall task supervision. All other procedures within the network are satellities, committed to performance of assignments made by the master procedure.

1. Initial Operation

Each input requiring a solution must contain a <command>, recognizable in the parsing algorithm as <verb> <fact>. Further, only the verbs PROVE and SHOW indicate a need for the problem-solving mechanisms; all other verbs initiate other procedures. Thus, when a <command> is parsed in the context of an <exercise>, the set of five pushdown stores are activated (see Figure III-E1) with bottom elements as markers. The desired and result is pushed onto the stores, along with the memory addresses of the figures involved. The simple variable, STK-PTR, marks the upper end of the stores for future reference.

FIGURE III-E1. GOAL STACKS



Shortly, the parsing algorithm completes the step which yields an <exercise>. The consolidation procedures perform their tasks (see Section III-D) and the analysis work is begun in earnest with the statement CALL PREDICTOR. The goal is examined to determine what relationship is desired and which structures are involved. The next step (actually a number of operations) is to examine the given information, sorting out the names and relationships involved.

Finally the plan of attack is developed. PREDICTOR examines the methods by which it could satisfy the main goal. A preferred order, dependent upon the available information, is established and execution of the plan is begun. A brief example will illustrate this process: PREDICTOR currently has only three available methods with which to establish the relation of congruence between two triangles: (side-angle-side, angle-side-angle and side-side-side). If the analysis reveals no known angles and one or more respective sides, the preferred order lists angle-side-angle as the last resort. Similarly, if two angles were known, then side-side-side would be the last method presented.

2. Proceeding Toward the Solution

Listed among the satellites mentioned earlier is the procedure which actually performs the labor involved in establishing a solution. Presented with a goal and a single method, ACCOMPLISH begins to work: establish the given by the suggested method, using all available information. The result returned to the master is a single selection from the list (SUCCESS, FAILURE, HELP).

The returned value of HELP is the signal that a subgoal has been generated and assistance from the heuristic section is required. More simply, the working procedure has proceeded to a point and discovered that additional information is necessary to complete the given task by the method assigned. This requirement is placed on the pushdown store by the satellite prior to returning the request for assistance. The analysis begins anew, examining the known information in terms of the new goal. Subsequently, the PREDICTOR examines its available methods and the complete cycle is reset.

The functional values of SUCCESS or FAILURE indicate the obvious results. The goal either was established or could not be established by the method given. Both results cause removal of the topmost elements of the pushdown stores and examination of the next goal. In the event that no goals remain (indicated by the markers), the final results are printed. If resolution of an additional goal is required, the process is repeated as above.

3. Communicating the Results

Regardless of the final outcome, GEO I provides printed output during each phase of its operation. A trace of this type is of obvious value in error detection. It further serves as a semblance of the communication which the human student might offer in the same situations.

If, in fact, the complete solution is derived, then a more formal result is presented, following completion of the trace steps. First, the problem is re-stated in precisely its

original format. The established proof is then presented as a list of arguments, each followed immediately by its defense.

IV. CONCLUSION

A. CURRENT CAPABILITIES

GEO I has not attained the ambitious goals which were established at the outset of the project. The system is not qualified to pass a final examination in Plane Geometry; nor does it exhibit any significant amount of intelligence. GEO I has, however, achieved a very modest level of success and has exhibited some of the essential characteristics necessary for achieving the goal. The machine has discovered the solutions for a number of elementary exercises, some of which are presented in the next pages. Additionally, the memory section retains a sizable information store, pertinent to the subject, which is retrievable upon command.

1. Exercise

An exercise (in a Plane Geometry course) is a specific application of a basic postulate or theorem. Derivation of a principle result is followed immediately by problems which employ that result (and others, recently derived). GEO I, in its present configuration, has obtained the correct proofs for a number of such exercises. Examples given in the following figures illustrate the machine solution of some simple problems.

Figure IV-A1 is subdivided into blocks marking the distinct phases of the machine process:

a. Input, Compilation and Data Storage Phases

Each input card is checked for errors and parsed into a complete statement by the compiler unit. Required data structures are established for the various entities as they are encountered. Recognition of the first statement of the next exercise triggers the data consolidation procedures.

b. Analysis and Problem-Solving Phases

The data analysis is accomplished and the main control routine is called to derive the method of attack. The resulting plan is to use assumption number 23 (i.e. "side-angle-side").

c. Proceeding Toward the Solution

The working procedures derive the correct proof and return their results to the main routine. The problem is restated and printed out with the appropriate steps.

d. The scratchpad memory is cleared, variables are reset and GEO I indicates that it is ready for the next exercise.

Figure IV-A4 represents the most interesting class of problems which GEO I is capable of solving at this time. The machine's study of the given information reveals that it is insufficient. The desired result is not immediate by direct application of any postulate or theorem. The analysis procedures, in working with the data structure, detect the fact that TRIANGLE ABD and TRIANGLE BCD have a common side, BD. Derviation of the single intermediate result was the key to the entire proof. The attack planning procedure selects the assumption for side-angle-side and the proof is immediate.

```
----------        PROBLEM SOLVING RUN FOR GEO I SYSTEM        ----------

                    PROBLEM: GIVEN: LINE BC EQU LINE EF.
                                    ANGLE BCA EQU ANGLE EFD.
                                    LINE AC EQU LINE DF.
                                PROVE TRIANGLE ABC CONGRUENT TRIANGLE DEF.
                  * PROBLEM: GIVEN: LINE BC EQU LINE EF.

********** BEGINNING WORK ON THE SOLUTION **********
THERE ARE 1 ANGLES AND 2 SIDES GIVEN.
TRYING SIDE-ANGLE-SIDE

      ********************     PROBLEM     ********************

                    PROBLEM: GIVEN: LINE BC EQU LINE EF.
                                    ANGLE BCA EQU ANGLE EFD.
                                    LINE AC EQU LINE DF.
                                PROVE TRIANGLE ABC CONGRUENT TRIANGLE DEF.

      ********************     PROOF     ********************

      STATEMENT 1.ANGLE ACB = ANGLE DFE
      REASON: GIVEN

      STATEMENT 2. LINE BC =  LINE EF
      REASON: GIVEN

      STATEMENT 3. LINE AC =  LINE DF
      REASON: GIVEN

      STATEMENT 4.TRIANGLE ABC CONGRUENT TRIANGLE DEF
      REASON: ASSUMPTION 23

**********     READY FOR NEXT EXERCISE     **********
```

A

B

C

D

FIGURE IV-A1 SAMPLE MACHINE SOLUTION

```
                    LINE AC EQU LINE DE.
                    LINE AB EQU LINE DE.
                PROVE TRIANGLE ABC CONGRUENT TRIANGLE DEF.
            *PROBLEM: GIVEN:TRIANGLE ABC AND TRIANGLE DEF.

********** BEGINNING WORK ON THE SOLUTION **********
THERE ARE 0 ANGLES AND 3 SIDES GIVEN.
TRYING SIDE-SIDE-SIDE

     ********************     PROBLEM     ********************

          * PROBLEM: GIVEN: LINE BC EQU LINE EF.
                    LINE AC EQU LINE DE.
                    LINE AB EQU LINE DE.
                PROVE TRIANGLE ABC CONGRUENT TRIANGLE DEF.

     ********************     PROOF     ********************

     STATEMENT 1. LINE AB =  LINE DE
     REASON: GIVEN

     STATEMENT 2. LINE BC =  LINE EF
     REASON: GIVEN

     STATEMENT 3. LINE AC =  LINE DF
     REASON: GIVEN

     STATEMENT 4.TRIANGLE ABC CONGRUENT TRIANGLE DEF
     REASON: ASSUMPTION 22

**********     READY FOR NEXT EXERCISE     **********
```

FIGURE IV-A2 SAMPLE MACHINE SOLUTION

```
                              ANGLE BAC EQU ANGLE EDF.
                              ANGLE ABC EQU ANGLE DEF.
                              LINE AB EQU LINE DE.
                         PROVE TRIANGLE ABC CONGRUENT TRIANGLE DEF.
                        * PROBLEM:GIVEN: ANGLE ABD EQU ANGLE BDC.

********** BEGINNING WORK ON THE SOLUTION **********
THERE ARE 2 ANGLES AND 1 SIDES GIVEN.
TRYING ANGLE-SIDE-ANGLE
BAC AND ABC HAVE THE COMMON SIDE AB
EDF AND DEF HAVE THE COMMON SIDE DE

     ********************     PROBLEM     ********************

               *PROBLEM: GIVEN:TRIANGLE ABC AND TRIANGLE DEF.
                         ANGLE BAC EQU ANGLE EDF.
                         ANGLE ABC EQU ANGLE DEF.
                         LINE AB EQU LINE DE.
                    PROVE TRIANGLE ABC CONGRUENT TRIANGLE DEF.

     ********************     PROOF     ********************

     STATEMENT 1.ANGLE ABC = ANGLE DEF
     REASON: GIVEN

     STATEMENT 2.ANGLE BAC = ANGLE EDF
     REASON: GIVEN

     STATEMENT 3. LINE AB =  LINE DE
     REASON: GIVEN

     STATEMENT 4.TRIANGLE ABC CONGRUENT TRIANGLE DEF
     REASON: ASSUMPTION 24

**********     READY FOR NEXT EXERCISE     **********
```

FIGURE IV-A3 SAMPLE MACHINE SOLUTION

```
                                        LINE AB EQU LINE CD.
                                    PROVE TRIANGLE ABD CONGRUENT TRIANGLE BCD.

********** BEGINNING WORK ON THE SOLUTION **********
THERE ARE 1 ANGLES AND 1 SIDES GIVEN.
TRYING SIDE-ANGLE-SIDE
ABD AND BCD HAVE THE COMMON SIDE BD

     **********          PROBLEM     **********

                  * PROBLEM:GIVEN: ANGLE ABD EQU ANGLE BDC.
                                   LINE AB EQU LINE CD.
                                   PPOVE TRIANGLE ABD CONGRUENT TRIANGLE BCD.

     **********          PROOF       **********

     STATEMENT 1.ANGLE ABD = ANGLE BDC
     REASON: GIVEN

     STATEMENT 2. LINE BD =  LINE BD
     REASON: IDENTITY

     STATEMENT 3. LINE AB =  LINE DC
     REASON: GIVEN

     STATEMENT 4.TRIANGLE ABD CONGRUENT TRIANGLE BCD
     REASON: ASSUMPTION 23

**********     READY FOR NEXT EXERCISE     **********
```

FIGURE IV-A4 SAMPLE MACHINE SOLUTION

It should be noted that such problems are typical of a first course in Plane Geometry, although the more complex exercises involve a number of small steps. Once the student has mastered the technique of locating intermediat results, he is able to solve a major portion of the problems with little difficulty (assuming he has a firm grasp of fundamentals).

2. Theorems

A theorem, for the purpose of this work, is a general statement which has been proved or conjectured. For example, "A tangent to a circle is perpendicular to the radius drawn to the point of contact." is a theorem. GEO I does not contain the procedures necessary for manipulation of such general statements. The system is limited to exercises which are specific statements of particular applications. Completion of the mechanics of problem-solving is, in the opinion of the author, a matter of some programming effort. Design of a machine which processes the general statements is then the most obvious area for further research.

3. Recitation

Recitation, as considered here, is the presentation of particular factual information in response to a given command. GEO I, through a simple dictionary search method, is capable of accessing 133 definitions, 43 basic assumptions, 93 theorems and 35 corollaries. Access permits both retrieval and modification of the stored information. An increase in the total amount of information available can easily be achieved by

a. increasing size of the one-dimensional storage arrays and

b. providing the desired information via the modification commands.

It should be pointed out because GEO I accepts only specific "by name" commands, such as "STATE THEOREM 10", the system cannot process general requests of the following nature: "State three theorems concerning right triangles"; "Give a postulate about the bisection of angles."

Figure IV-A5 presents a sample machine output containing examples of the various types of retrieval and modification. The marked sequence, a modification of THEOREM 11, deserves some explanation. The first two statements are the command input and the machine response. The single line, enclosed in single quotation marks, is the desired modification in string input form. The final two lines in the sequence are the repeat of the command input and the response with the

B. COMPILER STRUCTURE FOR PROBLEM-SOLVING PROGRAMS

The application of basic compiler structures in a problem-solving environment demands further investigation. GEO I has established that this unusual approach will provide results. The particular advantages of the system used in this work were discussed earlier. Other problem areas (e.g. trigonometry, analytic geometry, vector analysis) could be approached by the same methods with appropriate grammars and suitable procedures.

However, in selecting an under ying system one must consider the inherent assets and liabilities. The data structure of GEO I stands as a good example. It was constructed and manipulated at a level very close to assembly language. This

```
**********          TEXTBOOK OUTPUT          **********

    STATE DEFN OF 'RECTANGLE'.
RECTANGLE : A PARALLELOGRAM WITH AT LEAST ONE RIGHT ANGLE

  * WRITE DEFN 72.
SQUARE : A RECTANGLE WITH TWO ADJACENT SIDES EQUAL

  * STATE THEM 11.
11. IF TWO ANGLES HAVE THEIR SIDES RESPECTIVELY PARALLEL, THEY ARE EITHER
    EQUAL OR SUPPLEMENTARY.

  * 'THEOREM 11. ALL THINGS ARE EQUAL.'
*****     THEOREM 11 HAS BEEN MODIFIED.     *****

  * STATE THEM 11.
THEOREM 11. ALL THINGS ARE EQUAL.

  * STATE ASSUMP 22.
22. WHEN THREE SIDES OF ONE TRIANGLE ARE EQUAL RESPECTIVELY TO THREE SIDES
    OF ANOTHER TRIANGLE, THE TRIANGLES ARE CONGRUENT.

  * STATE ASSUMP 23.
23. WHEN TWO SIDES AND THE INCLUDED ANGLE OF ONE TRIANGLE ARE EQUAL RE-
    SPECTIVELY TO TWO SIDES AND THE INCLUDE ANGLE OF ANOTHER TRIANGLE, THE
    TRIANGLES ARE CONGRUENT.

  * STATE ASSUMP 24.
24. WHEN TWO ANGLES AND THE INCLUDED SIDE OF ONE TRIANGLE ARE EQUAL RE-
    SPECTIVELY TO TWO ANGLES AND THE INCLUDED SIDE OF ANOTHER TRIANGLE, THE
    TRIANGLES ARE CONGRUENT.

EOF EOF EOF
```

FIGURE IV-A5 COMMAND RESPONSE

tedious effort was undertaken only to secure the advantages of the compiler system. Other languages offered sophisticated data structures, but selecting any of those available entailed writing an entire compiler.

The development of problem-solving programs could be much easier if there were a system which provided (in addition to those benefits mentioned above):

1. function manipulation, string processing capabilities, sophisticated data structures and data handling processes, similar to those of LISP and PL/I;
2. general grammar analysis programs to detect the presence of such characteristics as ambiguity and double recursion;
3. a number of parsing methods, allowing the user to select the one best-suited to his application.

APPENDIX A

THE BNF DESCRIPTION OF PLANE GEOMETRY

```
<job>          ::= <input>

<input>        ::= <exercise>
                 | <input> * <exercise>
                 | <command>
                 | <input> * <command>
                 | <string>
                 | <input> * <string>

<exercise>     ::= <job type> : <job desc>

<command>      ::= <verb> <fact>.
                 | <verb> THEM <
                 | <verb> <defn abbr>.
                 | <verb> <assm abr>.

<assm abr>     ::= ASSUMP <number>
                 | ASSUMPTION_LIST

<defn abbr>    ::= DEFN <number>
                 | DEFN_OF <string>

<verb>         ::= ADD
                 | APPLY
                 | BISECT
                 | CONSTRUCT
                 | DEFINE
                 | DIVIDE
                 | DRAW
                 | EXTEND
                 | FIND
                 | JOIN
                 | LIST
                 | LOCATE
                 | MULTIPLY
                 | PROVE
                 | REMOVE
                 | REPLACE
                 | SHOW
                 | STATE
                 | SUBSTITUTE
                 | SUBTRACT
                 | TRISECT
                 | USE
                 | WRITE
```

```
<job type>      ::= THEOREM
                  | COROLLARY
                  | PROBLEM

<job desc>      ::= <given> <command>
                  | <given> <command> <hint>

<hint>          ::= HINT : <command>
                  | HINT : <given info>

<given>         ::= GIVEN : <given info>

<given info>    ::= <fact>.
                  | <fact> . <given info>

<fact>          ::= <l part> <relation> <l part> AT <entity>
                  | <l part> <relation> <l part>
                  | <l part>

<l part>        ::= <simple l>
                  | <simple l> <and part>
                  | <simple l> <with part>

<simple l>      ::= <entity>
                  | <entity> <poss part>
                  | <structure>
                  | <structure> <poss part>

<poss part>     ::= <poss verb> <number> <units>

<poss verb>     ::= IS
                  | HAS
                  | MEASURES

<units>         ::= INCHES
                  | DEGREES
                  | UNITS

<with part>     ::= WITH <simple l>
                  | WITH <simple l> <and part>

<and part>      ::= AND <simple l>
                    AND <simple l> <and part>

<entity>        ::= <point> <identifier>
                  | <line seg> <identifier>
                  | <angle> <identifier>

<point>         ::= POINT
                  | VERTEX
                  | ENDPOINT
                  | MIDPOINT
                  | CENTER
                  | INTERSECTION
```

```
<line seg>      ::= <strt seg>
                  | <curved seg>

<curved seg>    ::= ARC
                  | CURVE
                  | SEMICIRCLE

<strt seg>      ::= <line desc> <line type>
                  | <line type>

<line desc>     ::= VERT
                  | HORIZ

<line type>     ::= ALTITUDE
                  | BASE
                  | BISECTOR
                  | CHORD
                  | CIRCUMFERENCE
                  | DIAG
                  | DIAMETER
                  | HYPOT
                  | LEG
                  | LINE
                  | MEDIAN
                  | PERP
                  | RADIUS
                  | SECANT
                  | SIDE
                  | TANGENT
                  | TRANSVERSAL

<angle>         ::= <ang type 1> ANGLE
                  | <ang type 2> ANGLE
                  | ANGLE

<ang type 1>    ::= RIGHT
                  | ACUTE
                  | OBTUSE

<ang type 2>    ::= STRAIGHT
                  | REFLEX
                  | CENTRAL

<structure>     ::= <circle> <identifier>
                  | <polygon> <identifier>
                  | <triangle> <identifier>

<circle>        ::= CIRCLE

<polygon>       ::= <poly type> <poly name>
                  | <poly name>

<poly type>     ::= EQUIANGULAR
                  | EQUILATERAL
                  | REGULAR
```

```
<poly name>     ::= <quad name>
                  | PENTAGON
                  | HEXAGON
                  | HEPTAGON
                  | OCTAGON

<triangle>      ::= <tri type> TRIANGLE
                  | <poly type> TRIANGLE
                  | TRIANGLE

<tri type>      ::= <ang type 1>
                  | SCALENE
                  | ISOCELES

<quad name>     ::= QUAD_L
                  | <p gram>
                  | <t zoid>

<p gram>        ::= PARALLELOGRAM
                  | <rectangle>
                  | RHOMBUS

<rectangle>     ::= RECTANGLE
                  | SQUARE

<t zoid>        ::= TRAPEZOID
                  | ISOC_TRAP

<relation>      ::= <compare rel>
                  | <locate rel>
                  | <cond rel>
                  | <boolean>

<compare rel>   ::= EQU
                  | NEQ
                  | GTR
                  | LSS
                  | LEG
                  | GEQ
                  | SHORTER
                  | LONGER
                  | IDENTICAL
                  | PROPORTIONAL
                  | CONGRUENT
                  | SIMILAR
                  | REQUIDISTANT

<locate rel>    ::= ABOVE
                  | BELOW
                  | LEFT
                  | RT
                  | COMPLEMENTARY
                  | SUPPLEMENTARY
                  | ADJACENT
```

```
                |  OPPOSITE
                |  BETWEEN
                |  CIRCUMSCRIBED
                |  INSCRIBED
                |  CONCENTRIC

<cond rel>      ::= PERPENDICULAR
                |  PARALLEL
                |  INTERSECT
                |  COLINEAR
                |  <line desc>

<boolean>       ::= .and.
                |  .or.
                |  .not.
```

BIBLIOGRAPHY

1. McKeeman, W. M., Horning, J. J., and Wortman, D. B., A Compiler Generator, Prentice Hall, 1970.

2. Turing, A. M., "Computing Machinery and Intelligence" in Computers and Thought, edited by Feigenbaum, E. A. and Feldman, J., p.11-35, McGraw-Hill, 1963.

3. Minsky, M. L., and others, Semantic Information Processing, MIT Press, 1968.

4. Bobrow, D. G., "A Question-Answering System for High School Algebra Word Problems," in Proceedings AFIPS Fall Joint Computer Conference, p.591-614, 1964.

5. Evans, T. G., A Heuristic Program to Solve Geometric-Analogy Problems, Ph.D. Thesis, Massachusetts Institute of Technology, 1963.

6. Gelernter, H., "Realization of a Geometry Theorem-Proving Machine," in Proceedings of the International Conference on Information Processing, p.273-282, 1959

7. Gelernter, H. and Rochester, N., "Intelligent Behavior in Problem Solving Machines," IBM Journal of Research and Development, v.2, no. 4, p.336-345, 1958.

8. Gelernter, H., Hansen, J. R., and Loveland, D. W., "Empirical Explorations of the Geometry Theorem Machine," in Proceedings of the Western Joint Computer Conference, p.143-147, 1960.

9. IBM Research Report RC-281, System Requirements of a Digital Computer for the Manipulation of List Structures, by H. Gelernter, p.1-17, 1960.

10. IBM Research Report RC-282, A Manual for the Use of the FORTRAN-Compiled List Processing Language, by J. R. Hansen, p.1-36, 1960.

11. Leary, A. F. and Shuster, C. N., Plane Geometry, Charles Scribner's Sons, 1955.

12. Schnell, L. H. and Crawford, M. G., Plane Geometry, 3rd ed., McGraw-Hill, 1953.

13. Stabler, E. R., An Introduction to Mathematical Thought, p.11, Addison-Wesley, 1953.

14. Polya, G., How to Solve It, Princeton University Press, 1945.

15. Polya, G., Mathematics and Plausible Reasoning, v.1, Princeton University Press, 1954.

16. Polya, G., Mathematics and Plausible Reasoning, v.2, Princeton University Press, 1954.

17. Polya, G., Mathematical Discovery, v.1, John Wiley and Sons, 1962.

18. Polya, G., Mathematical Discovery, v.2, John Wiley and Sons, 1965.

19. Naur, P., et. al., "Report on the Algorithmic Language ALGOL 60," Communications of the Association for Computing Machinery, v.3, p.299, 1960.

20. Lee, J. A. N., The Anatomy of a Compiler, p.26-36, Reinhold, 1967.

21. Kildall, G. A., private communication.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documenation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Asst. Professor G. D. Gibbons Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	CAPT R. G. Osborne, USMC 840 Pinehurst Place Jackson, Mississippi 39205	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)

Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3 REPORT TITLE

GEO I: A Compiler Approach to Machine Problem-Solving

4 DESCRIPTIVE NOTES (Type of report and, inclusive dates)

Master's Thesis; June 1971

5. AUTHOR(S) (First name, middle initial, last name)

Ronald Glenn Osborne

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1971	76	21

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

The work described herein should be viewed as experimental research in the area of machine problem-solving. The essence of such study is the utilization of a digital computer for the discovery of problem solutions in regions which normally require the human faculty labelled intelligence. The domain of this effort was elementary Plane Geometry, including all of the assumptions, theorems and corollaries (and their associated exercises) normally considered in a first course. The ultimate goal was a machine which could attain a passing score on a final examination over the subject matter. The vehicle employed is a sizable computer program, designed and implemented under the facilities of a compiler generating system for execution on the the IBM System 360.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
BNF Grammar						
Compiler						
Dynamic Memory						

Thesis
0822
c.1

128408

Osborne
Geo I: A compiler approach to machine problem solving.